Evolution at the Sub-gene Level: Domain Rearrangements in the *Drosophila* Phylogeny – Supplementary Material

Yi-Chieh Wu, Matthew D. Rasmussen, Manolis Kellis

Supplementary Methods and Results

In this supplement, we let *architectures* refer to both a single architecture, e.g. a, ab, and a multiset (unordered list) of architectures, e.g. $\{a\}$, $\{ab\}$, $\{a, ab\}$, $\{a, a, ab\}$. Architectures can therefore be assigned to both genes (using the single architecture definition) and to entire species (using the multiset of architectures definition).

1 Modeling architecture rearrangements

Consider the problem of transforming a parent architecture to a child architecture. We represent these architectures as directed acyclic graphs, where the in-degree and out-degree of a node are each at most one. A module is indicated by a node, and neighboring modules within the same gene are joined with a directed edge. The problem is now equivalent to transforming from the parent graph to the child graph, where each allowable graph operation corresponds to an evolutionary event (Figure 1e):

- G: Add a node (corresponding to a module that does not currently exist).
- D: Duplicate a node.
- L: Remove a node.
- M: Add an edge between two (existing) nodes.
- S: Remove an edge between two (existing) nodes.

Note that these definitions require that generation, duplication, and loss occur at the module level. For example, generating a sequence of multiple modules is only possible through generation of the component modules. This assumption treats modules as the basic building blocks of a gene and implies that the generation, duplication, and loss of larger sequences (as measured by the number of modules) incur a higher cost.

Evolutionary events are applied in the following order: generation, duplication, loss, split, merge. There are also some caveats to account for the architecture representation. For example, we allow for the duplication and loss of entire architectures or sub-architectures, where the cost of duplication/loss corresponds to the number of modules duplicated/lost. Furthermore, the duplicated (sub-)architecture retains all the edges of the original, and edges within a lost (sub-)architecture can be removed without penalty. Duplication/loss of a sub-architecture may also incur one or two (hidden) splits, depending on whether the left and right end of the sub-architecture were connected to another module in the parent architecture; this prevents parental sub-architectures from appearing in isolation in the child without penalty. Finally, merges are free between two generated modules (but not between a generated module and an existing module); this allows for the generation of architectures in addition to the generation of modules.

Note that with this representation, there is a one-to-one mapping between nodes in the parent and child graphs (using the "null" node as the parent of a generated node and the child of a lost node). Tracing this mapping for each module in a reconstructed architecture scenario reveals the series of generations,

duplications, and losses that have led to the extant module counts, and tracing the addition or removal of edges across the mappings reveals the series of merges and splits that have led to the extant architectures.

To demonstrate the robustness of our architecture model, consider the case of a duplicated sub-architecture. For example, rearranging parent architecture $A = \{abc\}$ to form child architecture $B = \{ab, abc\}$ would infer a duplication of module a, a duplication of module b, and a (hidden) split between modules b and c. Note that (1) the original parent architecture abc is retained in B (with no inferred events), (2) no merge event is required between a and b since the multi-module sub-architecture ab arose from abc, and (3) a (hidden) split is inferred to account for the missing edge between b and c when ab is duplicated from abc.

2 Architecture scenario reconstruction

STAR-MP is a maximum parsimony algorithm for reconstructing the architecture scenario such that the total evolutionary cost is minimized. That is, for each architecture family, given the known species tree, the known architectures at the leaves, and the (inferred) module counts at the ancestors, STAR-MP infers the architectures at the ancestors and the events that occur along the branches. Here, we describe our architecture model and present the pseudocode for STAR-MP, which was described in brief in *Reconstructing architecture scenarios*. In the pseudocode, details may be omitted for clarity.

2.1 Notation

- T: the rooted, full, binary species tree, with nodes $N(T)$, edges $E(T)$, root $R(T)$, and leaves $L(T)$
- $n \in N$: a node with parent n_p and left and right children n_l and n_r
- A: an architecture with modules A^r
- $x \in \{G, D, L, M, S\}$: an evolutionary event with associated costs $cost(x)$
- $e = \{x_i\}$: a (multi)set of events
- $P(n)$: the modules at node n
- $Q(n) \in \{A_i\}, Q^*(n) = A$: the possible architectures, and the optimal (min-cost) architecture, at node n
- $E(A, B) = e, Cost(A, B) \in [0, \infty)$: the optimal set of events and the optimal cost to transform architecture B to architecture B
- $F(n, A)$: the optimal cost-to-go of assigning architecture A to node n, e.g. the optimal cost of events up to node n such that $Q^*(n) = A$
- $G(n, A)$: the traceback pointer for $F(n, A)$
- $H(n, n_l), H(n, n_r)$: the optimal events along edges $n \rightarrow n_l$ and $n \rightarrow n_r$

We also define the following operations on architectures: $A + B$ is the architecture containing the architectures in A and B, $A \cup B$ is the multiset of architectures that appear in either A or B, and $A \cap B$ is the multiset of architectures that appear in both A or B. For example, if $A = \{a, ab\}$ and $B = \{a, bc\}$, then $A+B = \{a, a, ab, bc\}$, $A \cup B = \{a, ab, bc\}$, and $A \cap B = \{a\}$. Similar operations will also be used for modules.

2.2 Pseudocode

Input: T, $cost(x)\ \forall x \in \{G, D, L, M, S\}$, $P(n)\ \forall n \in N(T)$, $Q(n)\ \forall n \in L(T)$
Output: The reconstructed architecture scenario, as defined by $Q^*(n), H(n, n_l), H(n, n_r)\ \forall n \in N(T)$

`/* Initialization */`
for $n \in L(T)$ **do**
  $F(n, Q(n)) = 0$
end

`/* Recursion */`
for $n \in N(T)$ *(post-order traversal) such that* $n \notin L(T)$ **do**
  `// Determine possible architectures`
  $Q(n) = \emptyset$
  for $B \in Q(n_l), C \in Q(n_r)$ **do**
    update $Q(n)$ with `archgen`$(P(n), B, C)$
  end
  `// Determine min-cost-to-go architecture`
  for $A \in Q(n)$ **do**
    `// Determine rearrangement cost`
    for $B \in Q(n_l) \cup Q(n_r)$ **do**
      $E(A, B), Cost(A, B) =$ `archeventcosts(`A,B,$\{cost\}$`)`
    end
    `// Find optimal`
    for $B \in Q(n_l), C \in Q(n_r)$ **do**
      $J(B, C) = F(n_l, B) + Cost(A, B) + F(n_r, C) + Cost(A, C)$
    end
    $G(n, A), F(n, A) = \arg\min$ / $\min_{B \in Q(n_l), C \in Q(n_r)} J(B, C)$
  end
end

`/* Termination */`
$Q^*(R(T)), F^* = \arg\min$ / $\min_{A \in Q(R(T))} F(R(T), A)$

`/* Traceback */`
for $n \in N(T)$ *(pre-order traversal) such that* $n \notin L(T)$ **do**
  $Q^*(n_l), Q^*(n_r) = G(n, Q^*(n))$
  $H(n, n_l), H(n, n_r) = E(Q^*(n), Q^*(n_l)), E(Q^*(n), Q^*(n_r))$
end

return $Q^*(n), H(n, n_l), H(n, n_r)\ \forall n \in N(T)$

Algorithm 1: `STAR-MP` **pseudocode.** Given the species tree, the ancestral module counts, and the extant architectures, `STAR-MP` reconstructs the architecture scenario by inferring the ancestral architectures and the events along the branches. Note that an explicit cost $cost(G)$ for generation is not required since modules are generated at the LCA of all species with the module. Code to account for ties is omitted for clarity; ties were broken arbitrarily in our implementation.

Input: set s of modules at the parent, children architectures B and C
Output: possible parent architectures

A = empty architecture
a = set of modules from s

/* Heuristic 1: If architecture *arch* appears in both children B and C and there are enough constituent modules in the parent, then that architecture must appear in the parent A. */
if *there are enough modules in s to cover all architectures in* $B \cap C$ **then**
  for *arch in* $B \cap C$ **do**
    add *arch* to A
    remove modules in *arch* from s
  end
end

if $s = \emptyset$ **then** // no parent modules left so return A
  yield A and stop iteration
end

/* Heuristic 2: If modules u and v are in both children B and C and there is no path $u \rightarrow v$ in either B or C, then there cannot be a path $u \rightarrow v$ in A. A path exists between any two modules u, v where u appears prior to v in the same gene. */
$paths = \emptyset$ // allowable paths in the parent
for $u \in a, v \in a$ *s.t.* $u \neq v$ **do**
  if $u \in B^r \cap C^r, v \in B^r \cap C^r$ **then**
    if $(u, v) \in$ *paths of* $B \vee (u, v) \in$ *paths of* C **then** add (u, v) to *paths*; // $u \rightarrow v$ exists in B or C
  else add (u, v) to *paths*; // at least one of u or v is missing from B or C
end

/* Heuristic 3: If both B and C contain u and v only in merged form uv, then u and v must be in merged form uv in at least one architecture of A. */
reqs = empty dictionary // required paths and whether the path is found
for $u \in a, v \in a$ *s.t.* $u \neq v$ **do**
  if $u \in B^r \wedge u \in C^r \wedge v \in B^r \wedge v \in C^r$ **then**
    if *u has 1 child* $\wedge$ *v has 1 parent* $\wedge$ *child of* $u = v \wedge$ *parent of* $v = u$ **then**
      // add requirement and whether it is satisfied by A
      if (u, v) *in* A **then** $reqs[u, v] = True$ **else** $reqs[u, v] = False$
    end
  end
end

/* Find allowable architectures using the remaining set of modules. For a set S, powersetorderedsubsets(S) is the set of all ordered subsets of S. */
aarchs = set of architectures from A // set of allowable parent architectures
for *arch* $\in$ powersetorderedsubsets($set(s)$) *s.t. arch is not empty* $\wedge$ *arch* $\notin$ *aarchs* **do**
  if $\exists (u, v) \in$ *arch s.t.* $(u, v) \notin$ *paths* **then** continue; // fails allowable paths
  add *arch* to *aarchs*
end

/* Now we have the allowable parent architectures so find the number of each architecture instance to satisfy the required parent modules. */
for *partition* P *of* s *s.t.* $\forall arch \in p, arch \in aarchs$ **do**
  // check if required merged modules are satisfied
  reqs_copy = copy of *reqs*
  for $arch \in P$, $(u, v) \in reqs$ *s.t.* $(u, v) \in arch$ **do** $reqs\_copy[u, v] = True$
  if *any reqs_copy failed* **then** continue

  // everything is satisfied
  yield $A + P$
end

Algorithm 2: archgen **pseudocode for generating possible architectures.** Given the available modules at the parent and the children architectures, archgen finds the possible parent architectures.

Input: parent architecture A, child architecture B, evolutionary costs *costs*
Output: the optimal series of events and the optimal cost required for the rearrangement

`/* Remove architectures that appear in both parent and child */`
for $arch \in A \cap B$ **do**
    `// do not remove if it is the last instance in the parent so it can be used in duplication`
    if *count of arch in* $A > 1$ **then**
        remove $arch$ from A and from B
    end
end

`/* Make graph representations */`
G_A, G_B = directed graph representations of A and B

`/* Generate/Duplicate/Lose modules through GDL, then Merge/Split modules through MS */`
E = empty list `// series of events required for rearrangement`
for $(E_{GDL}, G_{GDL}, gen, splits) \in$ `GDL(`G_a, G_b`)` **do**
    for $E_{MS} \in$ `MS(`$G_{GDL}, G_B, gen, splits$`)` **do**
        `// put GDL and MS events together and store`
        append $E_{GDL} + E_{MS}$ to E
    end
end

`/* Find optimum:` `findcost`$(e, costs)$ `determines the total cost of events` e `given costs` *costs* `*/`
$E^*, C^* = \arg\min / \min_{e \in E}$ `findcost(`e,*costs*`)`
return E^*, C^*

Algorithm 3: `archeventcosts` **pseudocode for finding the optimal events and cost for an architecture rearrangement.** Given a parent architecture, a child architecture, and the cost of evolutionary events corresponding to elementary graph operations, `archeventcosts` finds the optimal series of events and the optimal cost to transform the parent architecture to the child architecture. This is split into two parts: (1) generation, duplication, loss, and possibly split of modules until the parent and child architecture have the same number of modules, and (2) merge, split of modules until the parent architecture and child architecture are equivalent.

Input: parent architecture graph G_A, child architecture graph G_B
Output: series of events, resulting graphs, generated modules, and incurred splits for the rearrangement

`/* Generations - modules are generated (and then possibly duplicated) as single modules */`
E_G = empty list `// list of events due to generation (and duplication of generated modules)`
$gen = \emptyset$; $duplst, loselst$ = empty dictionaries `// generated module and dicts of [reg,# of dups/losses]`
for *reg* $\in$ *set of modules in* G_A *and* G_B **do**
  a, b = count of *reg* in G_A and G_B
  if $a = b$ **then** continue ; `// no change in module count`
  else if $a = 0$ **then** `// gen + dups`
    add *reg* to *gen*, add b instances of *reg* to G_A, append GEN of *reg* and $b-1$ DUP of *reg* to E_G
  else if $a < b$ **then** $duplst[reg] = b - a$; `// dups`
  else if $a > b$ **then** $loselst[reg] = a - b$; `// losses`
end

`// no dups of existing modules and no losses`
if *duplst is empty* $\wedge$ *loselst is empty* **then** **yield** $E_G, G_A, gen, \emptyset$ and stop iteration

`/* Duplications - one element for each combination of dups. Also keep track of duplication sources (map from duplicated child module to source parent module). */`
$E_D, G_D, dupmap$ = empty lists `// lists of events, graphs, and maps due to duplication`
if *duplst is empty* **then** append copy of E_G, copy of G_A, empty map to E_D, G_D, *dupmap* ; `// no dups`
else
  `// duplicate parent architectures until there are enough modules - Afterwards, there may be extra modules due to multi-module duplications. These will be lost in the Loss phase.`
  for *each unique combination todup of architectures to duplicate s.t. the counts in duplst are satisfied* **do**
    E, G, map = copy of E_G, copy of G_A, empty map; append E, G, map to $E_D, G_D, dupmap$
    for *arch* $\in$ *todup* **do** add *arch* to G, add DUP of *reg* ($\forall reg \in arch$) to E, update *map*
  end
end

`/* Losses - one element for each combination of dups/losses */`
for $E, G, dmap \in E_D, G_D, dupmap$ **do**
  recalculate *loselst* (as in Generation phase) `// recalc due to presence of extra duplicated modules`
  if *loselst is empty* **then** **yield** copy of E,copy of G,gen,$\emptyset$ and continue ; `// no losses`
  `// calculate losses for each combination of lost nodes`
  for *each unique combination tolose of modules to lose s.t. the counts in loselst are satisfied* **do**
    E_L, G_L = copy of E, copy of G
    `// remove modules`
    $splits = \emptyset$ `// count a split (from source) only once`
    for $u \in tolose$ **do**
      `// Consider` $t-u-v$ `and` $t_s-u_s-v_s$`, where` $n_s = dmap[n]$ `(e.g. dup source) if` n `arose from a dup and` $n_s = n$ `o.w. For each edge, remove the edge, and incur a split if both modules are not lost and the source of the edge was not already split.`
      if $\exists t$, remove (t, u) from G_L, add (t_s, u_s) to *splits*, append SPLIT of (t, u) to E_L if $(t_s, u_s) \notin splits$
      do same for (u, v) and (u_s, v_s)
      `// remove the module`
      remove u from G_L, append LOSS of u to E_L
    end
    `// remove redundancies (duplication and loss of same node)`
    for $u \in dmap \cap tolose$ **do** remove DUP of u and LOSS of u from E_L
    yield $E_L, G_L, gen, splits$
  end
end

Algorithm 4: `GDL` **pseudocode for rearranging architectures through generation/duplication/loss events.** Given a parent architecture and a child architecture, `GDL` finds the series of generation/duplication/loss events so that the resulting parent and child architectures have the same number of modules. Note that (1) which nodes are duplicated affects resulting architectures and possibly incurs extra splits and losses, and (2) which nodes are lost affects resulting architectures and possibly incurs extra splits; thus, `GDL` considers each possible combination of duplications and losses.

Input: parent architecture graph G_A, child architecture graph G_B, generated modules *gen*, incurred splits *splits* during generation/duplication/loss
Output: series of events required for the rearrangement

`/* Remove architectures that appear in both` G_A `and` G_B `*/`
for $arch \in G_A \cap G_B$ **do**
    remove $arch$ from G_A and G_B
end

`/* No merge/splits if no architectures left */`
if G_A *is empty* $\wedge$ G_B *is empty* **then**
    yield empty list and stop iteration
end

`/* Permute the nodes of one graph to look at all node-to-node assignments between graphs */`
for *pairwise assignment of nodes between* G_A *and* G_B *s.t. paired nodes have same module type* **do**
    `// add/remove edges (incur M/S events)`
    G_A^{copy} = copy of G_A
    E = empty list
    `// split (remove edge from` G_A^{copy}`) - must do first in case of insertion/deletion`
    for $(u_A, v_A) \in G_A$ **do**
        (u_B, v_B) = assignment of (u_a, v_a) from G_A to G_B
        if $(u_B, v_B) \notin G_B$ **then**
            remove (u_A, v_A) from G_A^{copy}
            `// do not penalize if this split was already incurred during duplication/loss`
            if $(u_A, v_A) \notin splits$ **then**
                (u, v) = module types corresponding to (u_A, v_A)
                append SPLIT of (u, v) to E
            end
        end
    end
    `// merge (add edge to` G_A^{copy}`)`
    for $(u_B, v_B) \in G_B$ **do**
        (u_A, v_A) = assignment of (u_B, v_B) from G_B to G_A
        if $(u_A, v_A) \notin G_A$ **then**
            add (u_A, v_A) to G_A^{copy}
            `// do not penalize if both` u `and` v `were generated`
            if $u \notin gen \vee v \notin gen$ **then**
                (u, v) = module types corresponding to (u_B, v_B)
                append MERGE of (u, v) to E
            end
        end
    end
    yield E
end

Algorithm 5: `MS` **pseudocode for rearranging architectures through merge/split events.** Given a parent architecture and a child architecture, `MS` finds the series of merge/split events so that the resulting parent and child architectures are equivalent.

3 Using domain annotations

As noted in the main text, the step of 'identifying modules and module families' in our phylogenomic pipeline may be replaced by a database search against existing domains (e.g. Pfam, SCOP, SMART, CDD, etc). We have chosen to use a *de novo* approach to module identification rather than using a domain database search for a number of reasons.

Our main reason is that we wished to make no *a priori* assumptions about the identity or boundaries of the modules. As mentioned in the main text, domain databases are often biased, for example, towards

domains with known structures or function. However, our definition of modules is evolutionarily-based and depends solely on sequence conservation.

An analysis of genome coverage (excluding singleton domains or modules) also revealed that only 62% of *Drosophila* genes have Pfam annotations compared to 82% of genes with module annotations. If we include singleton domains/modules, the change in coverage for Pfam annotations is negligible while the coverage for module annotations increases to 85%, with the remaining 15% of genes lacking BLAST hits that pass our filters. We believe that this difference in coverage is because our approach captures both known and *unknown* domains; in particular, it captures domains that are evolutionarily (rather structurally or functionally) conserved.

In addition, domain definitions are compiled using genomes across the three domains of life, meaning that domain families may be overclustered when looking at a small subset of genomes such as the ∼60 myr Drosophila clade. (Recall that gene and domain families are defined as the set of genes/domains that descend from a single gene/domain in the most recent common ancestor of all species under consideration. Therefore, restricting the genomes to a small subset will break the original families into many smaller clusters.) A major benefit of our approach is that it can be used at multiple timescales: we can look across the three domains of life as in domain databases, or we can find novel clade-specific domain families that may be missing from domain databases, as in our analysis of *Drosophila*. For comparison, ADDA found a number of novel domain families missing from Pfam and SMART, with the majority of these new families specific to a single domain of life. Such novel domain families may also be present within the *Drosophila* phylogeny (perhaps to confer clade-specific biological functions), and a such using domain definitions compiled across all three domains of life may lack the power to detect such recently evolved families.

Our approach can also capture known and unknown domains and neutral evolutionary events. In particular, we can identify modules linked to a protein function but associated with an unknown domain. This is important, as we are also interested in analyzing genome-wide event rates (or counts), and if we focus on the subset of genes in which the merged or split domain has a known function, these rates (counts) may be biased.

Finally, our approach has a higher power than a database search, and moreover, it can be applied to newly sequences genomes to discover new modules. In particular, we can analyze a group of closely related genomes that are together distant from other genomes. As mentioned previously, our method will find novel domain families that have evolved solely within the newly sequenced clade without requiring these families to be defined in domain databases.

4 Promiscuous modules

Analysis in our main text (*Results*) excluded promiscuous modules in our pipeline. Including these in our analysis decreased the number of architecture families to 14,156 (1.8% decrease), with 4201 families containing more than one module (0.03% decrease in ratio of # families with $\geq$ 2 modules/# families) and 4037 families containing a fusion or fission (<0.01% decrease in ratio). These "fusion/fission" families consist of 12,567 module families (0.2% increase) covering 46,100 sequences (0.2% increase) and involve at least one gene from 4533 (36.8%) of gene families (0.6% increase). As expected, the distribution of architecture families also shifted; for example, there would be 11 families with more than 20 modules and 22 families with more than 50 sequences (compare to Figure 6). Such increases are particularly problematic for our STAR-MP architecture reconstruction algorithm and would likely result in increased runtime or be too complicated for MP reconstruction.

5 Incorporating known rates of evolutionary events

We have incorporated known tendencies in event costs where applicable in our pipeline; for example, we used known estimated duplication and loss rates in *Drosophila* (Hahn et al. 2007) to reconstruct module phylogenies with SPIMAP. However, systematic studies of merge and split events (Snel et al. 2000, Kummerfeld and Teichmann 2005, Fong et al. 2007) have only determined total counts or merge-to-split ratios, and these are neither specific to the *Drosophila* clade nor do they incorporate architecture counts. The few studies on fusion and fission in *Drosophila* (Zhou et al. 2008, Rogers et al. 2009) focus on a subset of species and on

genes with significant experimental evidence. In contrast, it takes a systematic, genome-wide approach to determine event rates that are unbiased and reflective of the entire genome.

6 Cost of evolutionary events

We analyzed a subset of 200 families using varying event costs. Note that the cost of generation does not affect STAR-MP reconstructions since each module is assumed to have been generated only once at the most recent common ancestor of all species that contain the module. In addition, in our current implementation, duplication and loss costs also have limited effect since ancestral counts are inferred using the reconstructed module phylogenies, and a module that is duplicated and subsequently lost along the same branch does not incur any cost.

We tested six settings for the event costs: one in which all events were equal (as in the main text), four in which a single event (D,L,M,S) had twice the cost of the others, and one in which merges and splits were twice the cost of generations, duplications and losses. For each setting, we summed the number of inferred evolutionary events of each type, then computed its deviation from the inferred counts under equal costs. All deviations were less than 3.7% except for four cases: when merges had double the cost, the number of merges decreased by 23.9% and the number of splits increased by 20.3%, and when splits had double the cost, the number of merges increased by 36.6% and the number of splits decreased by 26.8%. Furthermore, while the number of inferred regions are the same across all settings (since we have used the same input module trees), the number of genes for a doubled merge cost decreased by 2.4% and for a doubled split cost increased by 2.9%. This is consistent with our expectations, as a higher merge cost should result in a larger number of merged ancestral genes (e.g. fewer genes given the same number of modules) so that fewer merges and more splits are inferred. Similarly, a higher split cost should result in a larger number of split ancestral genes (e.g. more genes given the same number of modules) so that more merges and fewer splits are inferred.

However, in this smaller set of families, each reconstruction contributes a larger portion to the total number of events; thus, many deviations could be attributed to the small number of families that have multiple maximum parsimonious reconstructions. (Remember that ties are broken randomly.) If we consider the 143 families for which only a single maximum parsimonious reconstruction exists for every setting, almost all deviations drop two-fold or more. For this filtered set, all deviations were less than 1.1% except for two cases: when splits have double the cost, the number of merges increased by 22.0% and the number of splits decreased by 9.1%. Note that a doubled merge cost negligibly affects the inferred evolutionary events, and that the deviations in merge and split counts for a doubled split cost have dropped. We believe that many of the deviations for the doubled split cost are due to cases in which an architecture is partially lost. Here, the high split cost causes STAR-MP to infer a split ancestral gene when other parameter settings would infer a merged ancestral form. (See also Supplemental Section 7.3.)

Our analysis shows that in almost all cases, the balance of inferred events is maintained since these events are constrained by the evolutionary evidence. Significant deviations may be seen if a higher split cost is used, but then, a larger number of merges and a lower number of splits will be inferred, which would further support our findings that merges are more prevalent than splits (*Common trends in architecture scenarios revealed by STAR-MP reconstruction*).

7 Validation

7.1 Detection of undercollapsed scaffolds

The BLASTp hits of a species versus itself were filtered to retain hits between genes located on different scaffolds and with percent identity $\geq 95\%$. These were run through LALIGN using nucleotide sequences extended to $\pm$ 2000 nt upstream and downstream, and hits with percent identity $\geq 98\%$ were retained. An architecture family is said to contain possibly undercollapsed scaffolds if at least two genes within the family have a hit in this final list.

7.2 EST and mRNA-seq evidence

ESTs were obtained from GenBank and compared against the protein sequences using BLASTx. Only hits with long ESTs ($\geq$ 250 nt), *e*-value $\leq 1 \times 10^{-5}$, percent identity $\geq$ 96%, and alignment length $\geq$ 50 aa were retained. If a single EST aligned to the same sequence in multiple places, the alignment with the highest percent identity was retained. An EST is said to span two genes if it aligns with both genes and the alignments are in the same direction and overlap by $\leq$ 15 aa.

mRNA-seqs at 36 and 75 nt resolution were obtained from modENCODE (`http://www.modencode.org`). Briefly, this protocol used polyA RNA extracted from *D. melanogaster*, *D. pseudoobscura*, and *D. mojavenis* male and female heads, with sequences aligned to the genome with Bowtie allowing for up to two mismatches. We mapped the mRNA to genes based on genome location, and an mRNA-seq was said to span two genes if it aligns with both genes and the alignments are in the same direction and do not overlap.

7.3 Simulations under various event rates

Keeping the generation rate constant at 1X the estimated true rate, we set the duplication, loss, merge, and split rates at 1X, 2X, and 4X the estimated true rates. We tested five different settings, simulating 1000 architecture scenarios for each setting (Supplemental Figure S2).

In general, STAR-MP has higher precision than sensitivity for any given event, and performance tends to degrade as the event rates increase and the true architecture scenarios become more complicated. Indeed, part of the decrease in sensitivity can be attributed to trying to explain more complex architecture scenarios with a conservative MP algorithm. STAR-MP also tends to have higher generation, duplication, and loss performance than merge and split performance, and as expected, generation, duplication, and loss performance is consistent across various merge and split rates. Interestingly, merge performance is typically higher than split performance. Further investigation showed that low split performance can be attributed to cases in which an architecture is partially lost. Here, the true reconstruction is a merged parent architecture undergoing a split and loss to result in the surviving sub-architecture; however, STAR-MP tended to reconstruct a split parent architecture so that only a loss is needed to produce the surviving sub-architecture.

7.4 Support for fusion and fission events using transcript evidence

We excluded from this analysis the 6.1% (249/4107) of merge/split scenarios have no merge or split events. This occured since we determined merge/split families based solely on the clustering of architectures into architecture families. However, all merge and split events in the family may have occurred prior to the species tree root; such a case can only be determined after the architecture scenario reconstruction.

Consider a merge or split event between two modules, as found by our architecture scenario reconstruction algorithm. Each of these events bifurcates the leaves, with one subset containing the leaves belonging in the subtree descended from the event, and the other subset consisting of the the rest of the leaves. We denoted these sets as the *merged genes* or *split genes*, respectively, if we were considering a merge event, or vice-versa if we were considering a split event. A merged gene was classified as *consistent* if there existed at least one EST/mRNA-seq that covers the boundary between the modules, and as *unknown* otherwise. (Note that we did not allow a merged gene to be *inconsistent* with the evidence.) A split gene was classified as *inconsistent* if there existed at least one spanning EST (mRNA-seq) (Supplemental Section 7.2) for the gene, as *consistent* if there existed an EST but no spanning EST (mRNA-seq), and as *unknown* otherwise. Once the genes were classified, an event was classified as *consistent* if there existed at least one *consistent* merged gene and at least one *consistent* split gene, as *inconsistent* if there existed at least one *inconsistent* split gene, and as *unknown* otherwise. Finally, the events for each scenario were pooled, and each scenario was classified as *inconsistent* if any event was *inconsistent*, as *consistent* if all events were *consistent*, and as *unknown* otherwise.

8 No substitution rate bias in merge/split families

Analysis of *Adh*-derived chimeric genes previously revealed elevated rates of amino acid substitution after merge events (Long and Langley 1993, Jones et al. 2005, Jones and Begun 2005). It was speculated that

the new function of the chimeric gene no longer required strong conservation, or that amino acids along the merge boundary rapidly evolved to repair any possible damage incurred by the merge event. To examine whether this bias occurs at a systematic level, we computed the substitution rate for each module family (Supplemental Figure S5) using the SPIMAP model of assigning a gene-specific (here module-specific) rate and species-specific rate to each tree (Rasmussen and Kellis 2007; 2010). Notice that while the distributions are significantly different, the effect size is small (fold of median rates = 0.970–0.975). Furthermore, contrary to the results with *Adh*, we found that modules in merge/split families tend to have *lower* substitution rates than average. We believe that this discrepancy may be attributed to limitations in our model. For example, we computed rates across entire module trees, but it may be more appropriate to compute separate rates for portions of the tree affected and not affected by the merge/split event. We also did not consider differences in composition across species, which may confound the gene-specific and species-specific rate. Finally, we question whether an elevated substitution rate is indeed expected, since it is also plausible that modules should be more conserved after merge/split events in order to maintain functionality; that is, modules that undergo many substitutions may lose functionality and degenerate into pseudogenes.

9 GO term and Pfam domain enrichment/depletion

Enrichment/depletion values were computed using GOseq (Young et al. 2010) to correct for possible length (Supplemental Figure S4) and substitution bias (Supplemental Section 8, Supplemental Figure S5) in the data. Briefly, GOseq determines a probability weighting function that quantifies how the probability of a gene selected out of the reference set changes as a function of some external variable such as transcript length. It then resamples the genes many times, weighting the probability of choosing a gene using this function, and uses the resulting sampling distribution to calculate a p-value. Alternatively, GOseq uses the Wallenius non-central hypergeometric distribution to approximate p-values; we use this approximation in our analysis. p-values were corrected separately for length bias and substitution bias; correction for substitution bias did not change the set of enriched/depleted terms, so only the correction for length bias is shown in the main text.

For gene functions, we looked at gene ontology (GO) annotations for *D. melanogaster*, as the other species have little to no GO annotation. Only GO terms with experimental evidence were retained, and a gene with a GO term was expanded to also include all parent GO terms. *D. melanogaster* genes contain 4524 unique GO terms, 3327 of which are found in genes that participate in architecture families with merges or splits. Enrichment/depletion values were computed separately for each of the three ontologies biological process, cellular component, and molecular function. For domains, we looked at Pfam domains for all species. The *Drosophila* clade contains 3204 unique Pfam domains, 1510 of which are found in genes that participate in architecture families with merges or splits. After correcting for length bias, no Pfam domains were significantly enriched or depleted (hypergeometric test, $p < 0.001$, FDR correction).

10 Functional complementarity using DroID database

Rather than looking at shared GO terms, we searched for gene partners against the *Drosophila* interactions database (DroID) (Yu et al. 2008) (April 2010 release). Of the 1222 gene partners, 589 are those in which both genes have at least one known PPI (but not necessary with each other) and 135 (22.9% of 589) are those in which the genes are known to interact with each other. This is compared to 0.3% random (fold = 75.74, $p < 0.001$).

Using a set of high confidence interactions in which we retain only PPI with experimental evidence (discarding those detected through homology), these numbers reduced to 57 gene partners in which both genes have at least one known PPI, 24 (42.1%) in which the genes are known to interact with each other, and 0.38% random (fold = 110.53, $p < 0.001$). However, low counts means that we must take care in making any biological statements.

11 PPI in conservative set of architecture families

Using the conservative set of architecture families, we identified 222 gene partners within *D. melanogaster*. Of these, 33 gene partners have both genes annotated with GO terms, and 30 (90.9% of 33) share at least one GO term, compared to 61.8% random (fold = 1.47, $p < 0.001$). Using the DroID database, 140 gene partners are those in which both genes have at least one known PPI and 48 (34.3%) are those in which the genes are known to interact with each other, compared to 0.3% random (fold = 107.05, $p < 0.001$). Using the set of high confidence interactions, these numbers reduced to 16 gene partners in which both genes have at least one known PPI, 5 (31.3%) in which the genes are known to interact with each other, and 0.37% random (fold = 83.48, $p < 0.001$).

12 Detection of frameshift mutations

To investigate how often nucleotide similarity is not propagated to peptide similarity, we ran pairwise all-vs-all tBLASTx between the species, then post-processed the alignments with LALIGN and filters (e-value $\leq 1 \times 10^{-5}$ and percent identity $\geq 80\%$; a higher threshold for percent identity was used to account for the smaller nucleotide alphabet). We found 2740 hits at the nucleotide level but not at the peptide level, 130,422 hits at both the nucleotide and peptide level, and 345,554 hits at the peptide level but not at the nucleotide level. This translated to 1429 additional genes that may participate in a merge/split architecture family; note that the actual number of additional genes that participate in merge/split families is likely smaller than this count. For example, some of the genes may be part of families in which each gene consists of a single module, and the module is simply frameshifted in some genes, or some of the frameshift mutations may be a result of frameshift sequencing errors.

13 Systematic detection of gene fusion and fission by mechanism

13.1 Fusion/fission of adjacent genes

Two modules that merge or split were tagged if they were found in neighboring genes and the modules would be adjacent if the neighboring genes were considered as a single gene. This list was expanded to genes by looking for all genes with these tagged modules and including only those genes that are neighboring or are found in species descended from the most ancestral branch with a merge or split of the modules. Note that this list of genes includes both parental and children genes (e.g. pre- and post-merge/split genes), as this allows for ambiguities that may have arisen from ties in the MP reconstruction (e.g. one MP reconstruction finds split genes at the root followed by a merge along one branch, whereas another reconstruction finds a merged gene at the root followed by a split along the other branch). We also tested for experimental support for each gene by looking at EST and mRNA-seq data. Using our previously determined set of spanning ESTs (mRNA-seqs) (Supplemental Section 7.2), we called a gene *consistent* if all of the associated fragmented genes had EST (mRNA-seq) evidence but none were part of a spanning EST (mRNA-seq), *inconsistent* if at least one of the associated fragmented genes was part of a spanning EST (mRNA-seq), and *unknown* otherwise.

Genes that may have resulted from large-loop mismatch repair and replication slippage were detected by looking for merged genes flanked by (but not necessarily next to) two genes, with one gene containing one of the merged modules and the other gene containing the other merged module.

13.2 Retrotransposition and exon shuffling

While retrotransposition and exon shuffling are two separate mechanisms for novel gene formation, it has been suggested that retrotransposition is a driving mechanism for mediating exon shuffling (Gilbert et al. 1995), and exon shuffling by retrotransposition is one method for conferring novel gene functions to the resulting chimeric gene rather than allowing the retrosequence to degenerate into a psuedogene (Long 2001). To find retrotransposed-mediated exon shuffling events, we searched for modules that undergo a merge or split and that have multiple exons in at least one gene with merged form and a single exon in at least one gene with split form. This ignores possible retrotranspositions of single-exon genes which cannot be distinguished

in our analysis from simple duplication. Exon comparison was performed at the module level to allow for (1) a chimeric gene to result from a multi-exon parent fusing with a second retrotransposed parent, and (2) partial gene retrotransposition. Furthermore, each merge or split formed a bifurcation of the species tree, and the merged form and split form must belong to different subtrees formed from this bifurcation. Only genes where the module has a single exon were included in the final count.

13.3 Duplication-degeneration

We constructed syntenic modules by defining a syntenic block to be at least three genes within 200 kb of each other with no other blocks in between. Two genes are the result of duplication-degeneration if (1) they result from a split, and (2) they belong to different syntenic blocks but have hits to the same syntenic block. False detection could occur due to faulty syntenic blocks. For example, a missed connection between two scaffolds may result in one syntenic block being separated into two blocks. As a simple test, we checked whether both genes are located at the ends of their respective scaffolds; 36 modules and 50 genes remain. Note that this method of detection does not take into account the case of undercollapsed assemblies where genes are located within the scaffolds.

Supplementary Figures and Tables

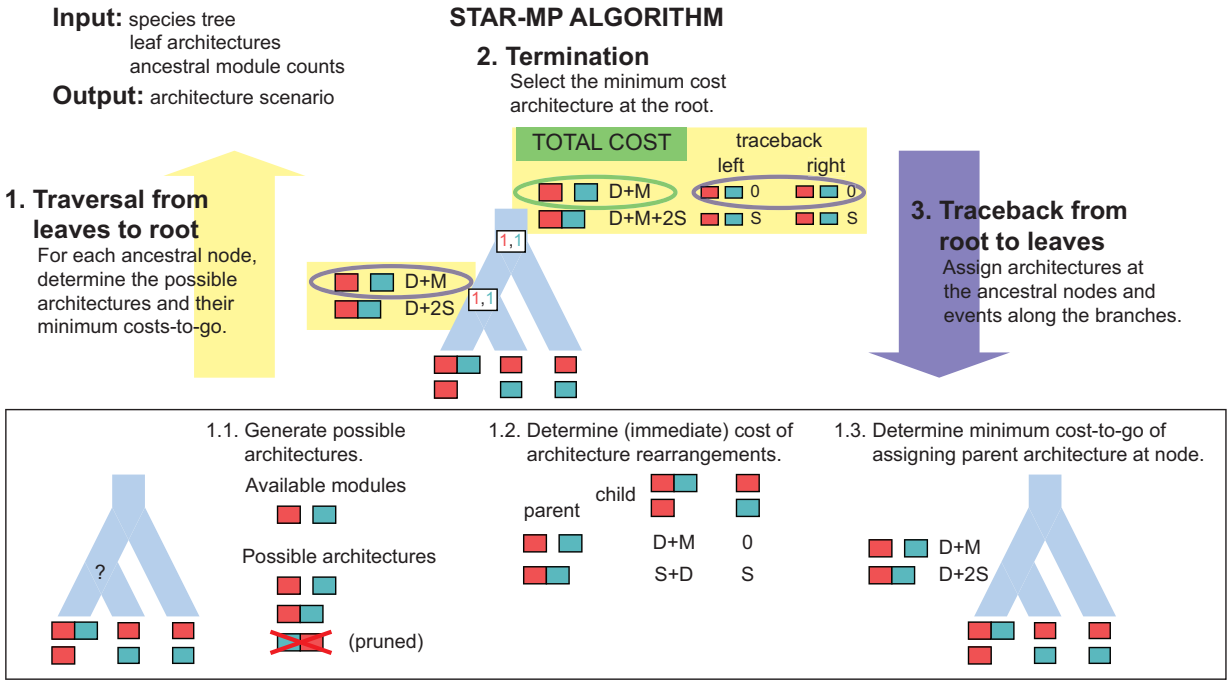

Figure S1: STAR-MP algorithm. The forward phase of the STAR-MP algorithm performs a post-order traversal of the tree (step 1), considering at each stage a triplet of one parent node and two child nodes. The available parent modules (provided as input and found in our pipeline through reconciled module trees) and the possible children architectures (provided as input if the child is a leaf node or found recursively if the child is an ancestral node) are known. The possible parent architectures are generated through set permutations and pruned heuristically, and the minimum costs-to-go of assigning a parent architecture is determined (see inset at bottom, where the parent node under question is denoted by the '?'). The children architectures and events along the branches that led to this minimum cost-to-go is also retained for later traceback. This is repeated until the root of the tree is reached (step 2), at which point the minimum cost architecture is assigned to the root. The backward phase then performs a pre-order traversal of the tree (step 3) to assign the ancestral architectures and events.

input rate multipliers				actual rate multipliers			
D	L	M	S	D	L	M	S
1	1	1	1	0.98	0.94	0.98	0.97
1	1	2	2	0.98	0.97	1.40	1.39
1	1	4	4	0.98	0.98	1.97	1.97
2	2	2	2	1.74	1.57	1.23	1.27
4	4	4	4	3.23	2.61	1.66	1.74

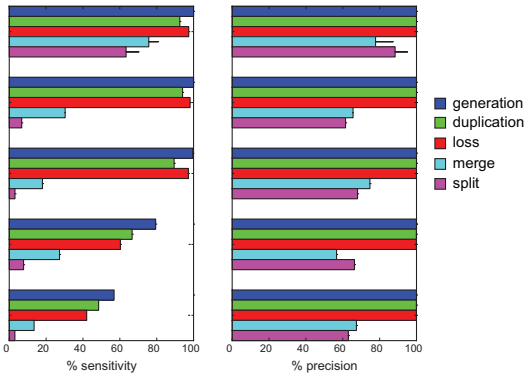

Figure S2: Sensitivity and precision of STAR-MP in simulation under various event rates. See Figure 4 for details. Sensitivity decays dramatically as event rates increase, as is consistent with a conservative MP reconstruction. In contrast, precision is robust to event rates. Event rate multipliers were obtained from the simulations and differ from the input rate multipliers of N: (1,1,1,1), MS2: (1,1,2,2), MS4: (1,1,4,4), all2: (2,2,2,2), and all4: (4,4,4,4) due to our approach of discarding events that were impossible with the given starting architecture.

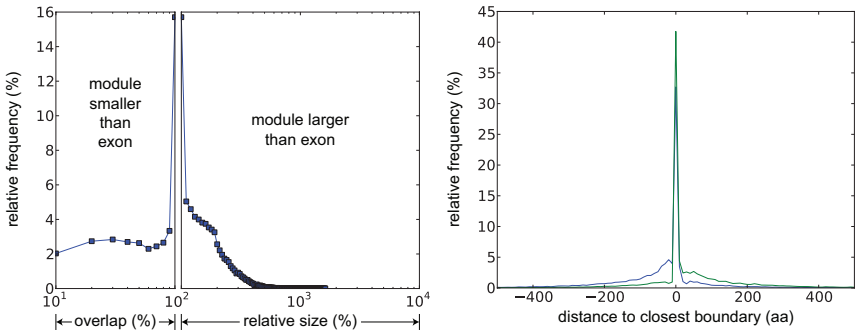

Figure S3: Correlation of module and exon boundaries. See Figure 5 for details.

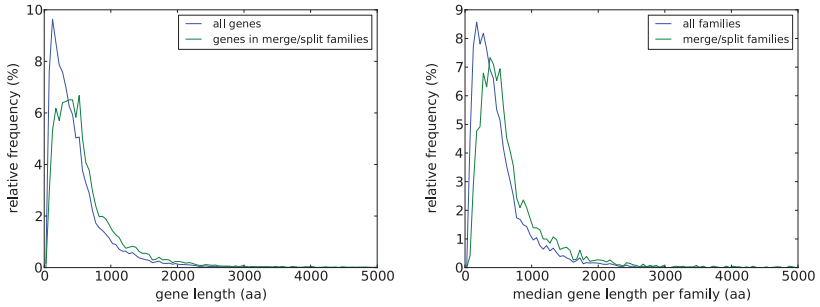

Figure S4: Length bias. Genes in merge/split families tend to be longer than average. (*Left*) The distribution of gene lengths, bin size = 50 aa. Median lengths for all genes and for genes in merge/split families were 361 aa and 492 aa, respectively; the distributions differed significantly (Mann-Whitney U = 3.22×10^9, $p < 2.23 \times 10^{-308}$, one-sided). (*right*), The distribution of median gene lengths per architecture family, bin size = 50 aa. Median lengths for all families and for merge/split families were 391 aa and 524 aa, respectively; the distributions differed significantly (Mann-Whitney U = 3.19×10^7, $p = 9.23 \times 10^{-116}$, one-sided).

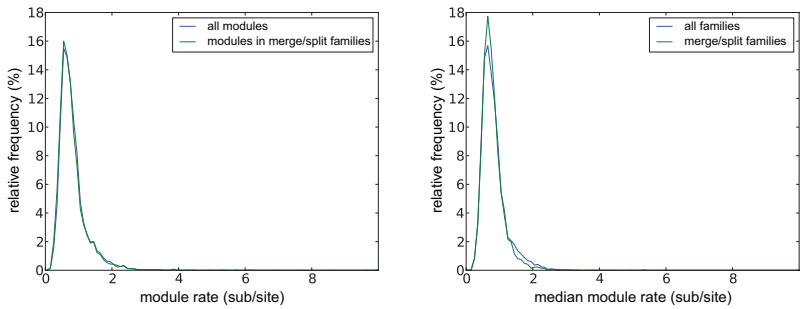

Figure S5: No substitution rate bias. Genes in merge/split families have similar substitution rates compared to average. (*Left*), The distribution of substitution rates across module families, bin size = 0.1 sub/site. Median rates for all modules and for modules in merge/split families were 0.723 sub/site and 0.701 sub/site, respectively; the distributions differed significantly (Mann-Whitney U = 1.35×10^8, $p = 5.23 \times 10^{-10}$, one-sided). (*right*), The distribution of median substitution rates per architecture family, bin size = 50 aa. Median rates for all families and for merge/split families were 0.744 sub/site and 0.725 sub/site, respectively; the distributions differed significantly (Mann-Whitney U = 2.85×10^7, $p = 1.73 \times 10^{-4}$, one-sided).

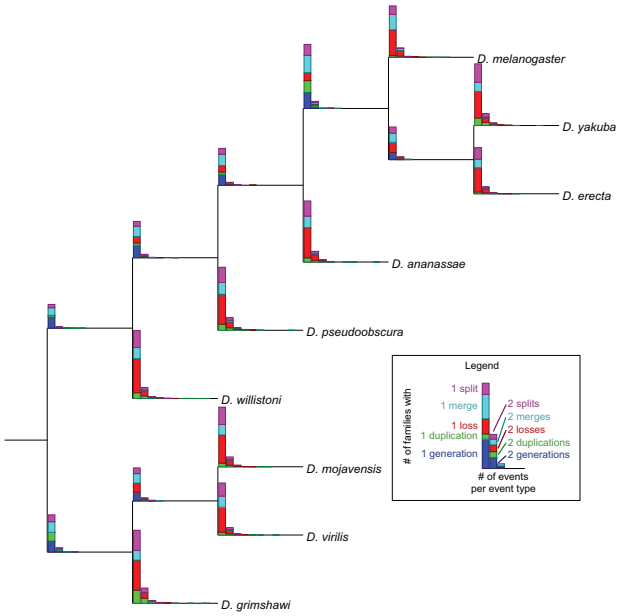

Figure S6: Distribution of inferred event counts across architecture families. Event counts are not dominated by a subset of families. Most families have at most one event per type along each branch.

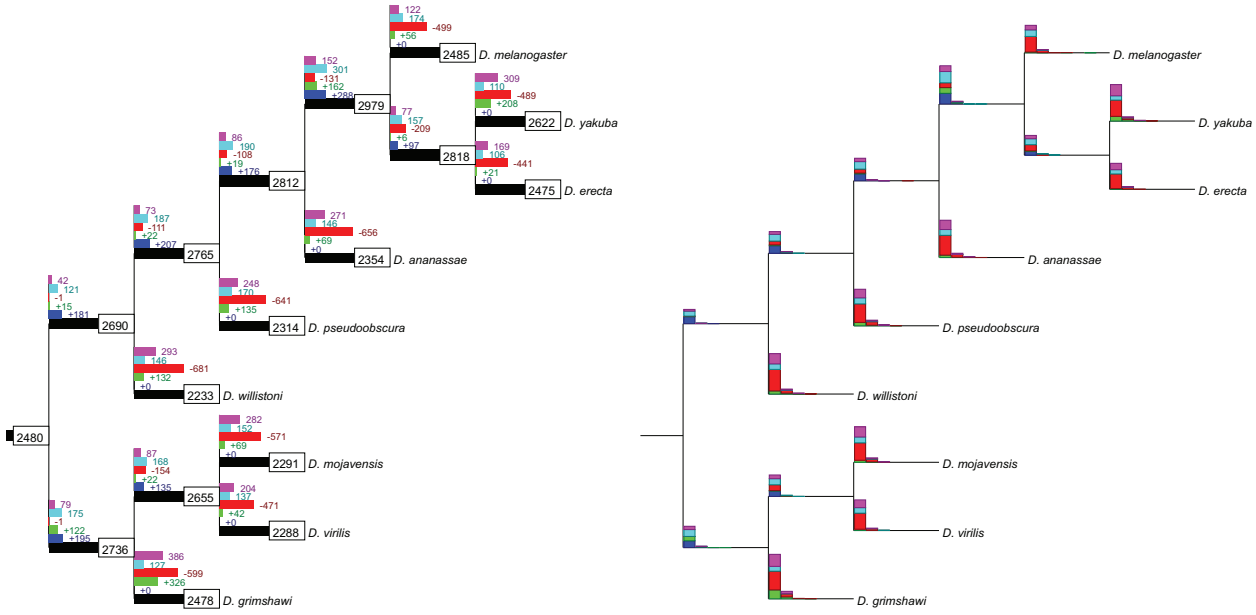

Figure S7: Distribution of inferred evolutionary events using a conservative set of architecture families. See Figures 7 and S6 for details.

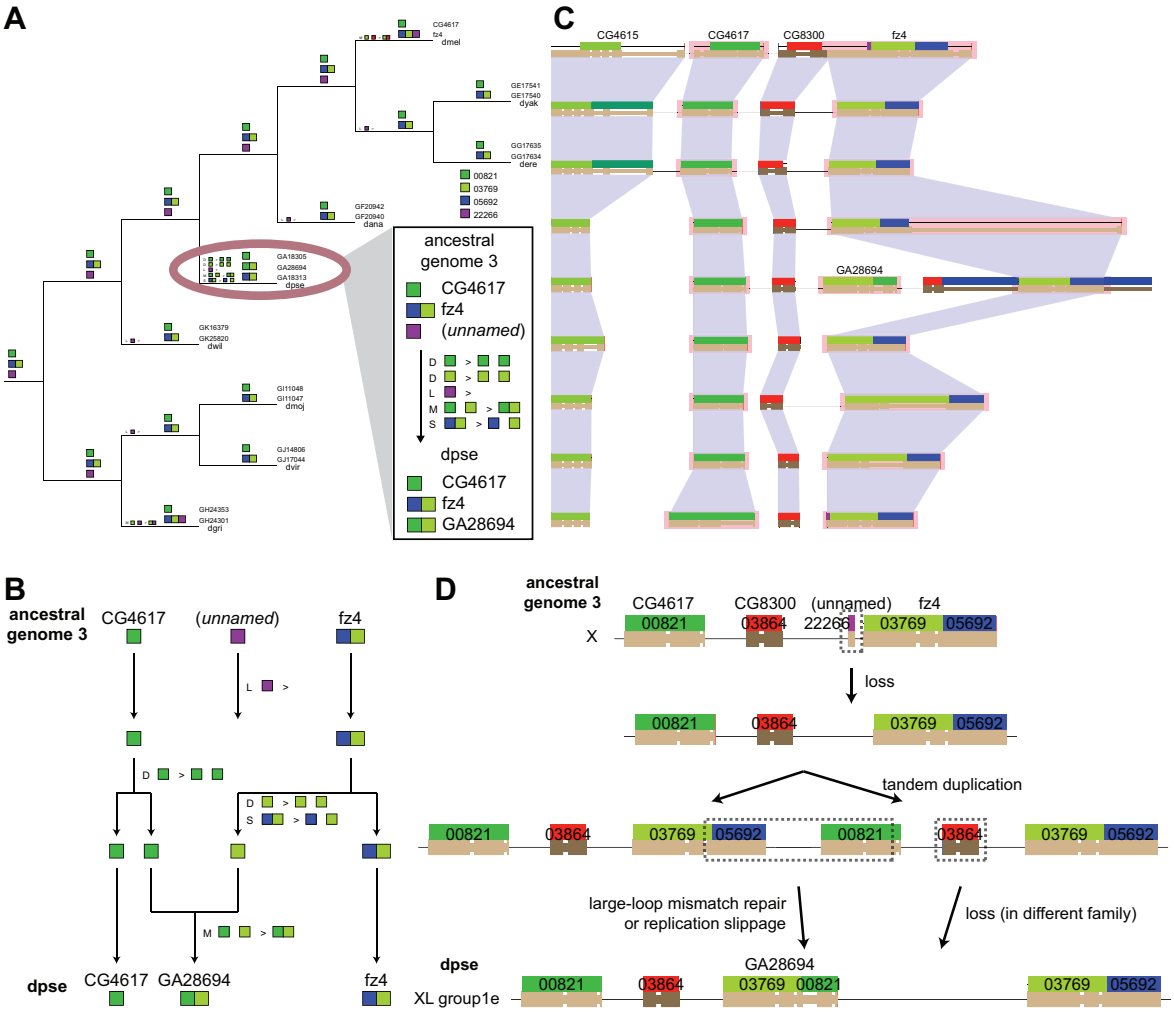

Figure S8: The inferred evolutionary history of *GA28694* in *D. pseudoobscura* from *CG4617* and *frizzled 4* (*fz4*). (*A*) The MP architecture scenario. Architectures 00821 and 05692-03769 are usually in separate genes corresponding to *CG4617* and *fz4* orthologs. However, *dpse* contains the fused architecture 00821-03769 in gene *GA28694*. (*B*), Along the *dpse* branch, the MP reconstruction infers the loss of module 22266 and duplications of modules 00821 and 03769 followed by a merge to form 00821-03769. A (hidden) split is also inferred between 05692 and 03769 due to the sub-architecture duplication of 03769 from *fz4* (05692-03769). (*C*) A genome level view shows that the orthologs of *CG4617* and *fz4* are adjacent in all species except *dpse*. In *dpse*, the new, fused gene *GA28694* is located between *CG4617* and *fz4*. (*D*) The inferred evolutionary history of *GA28694* in *dpse* through duplication and exon shuffling. We postulate that *GA28694* was formed by tandem duplication of the chromosomal region from *CG4617* to *fz4*, followed by either large-loop mismatch repair or replication slippage to form the merged gene *GA28694*. Duplication allows *dpse* to retain the original gene functions of *CG4617* and *fz4* and gain a new function with *GA28694*. Note that while module 05692 is duplicated as part of the tandem duplication, it is subsequently lost in *dpse*; thus, the MP reconstruction does not infer a duplication of 05692.

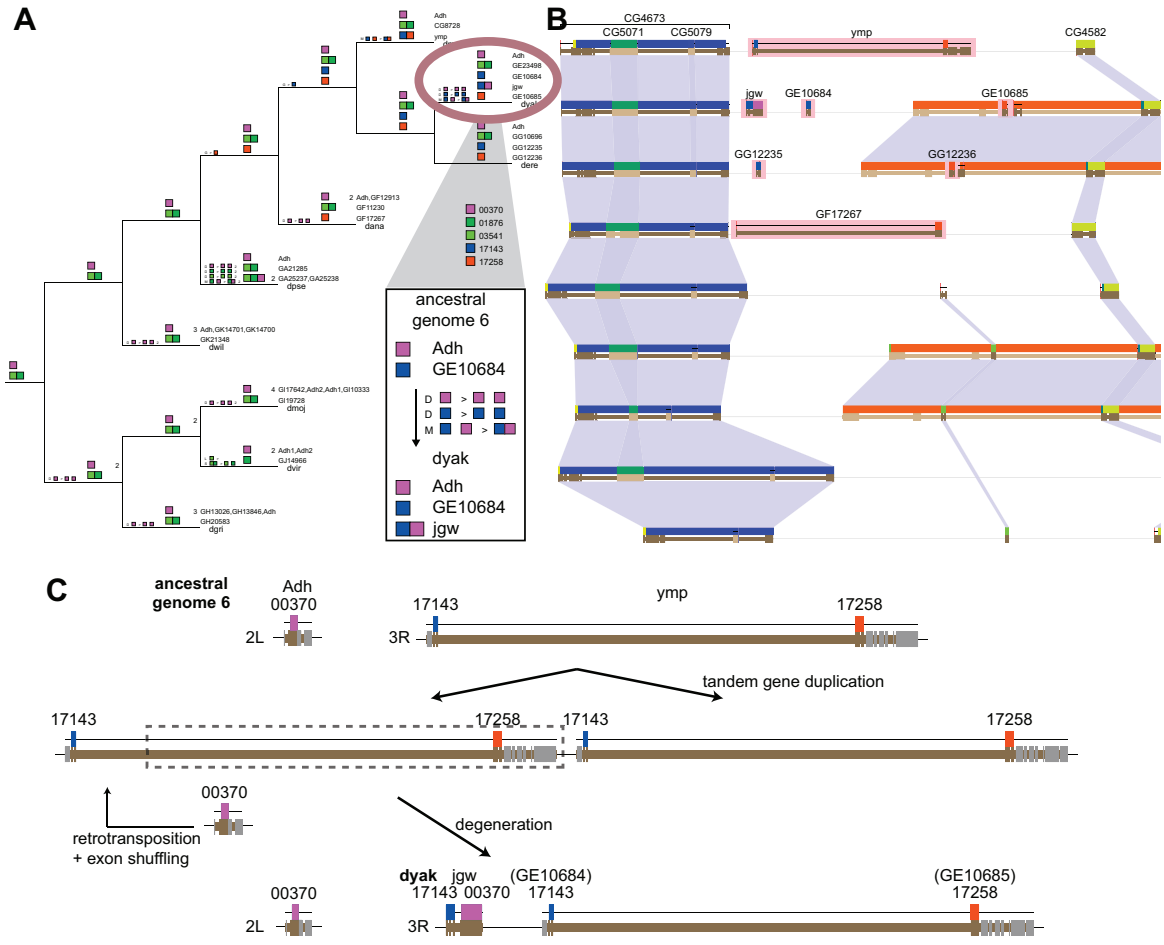

Figure S9: The inferred evolutionary history of *jingwei* (*jgw*) in *D. yakuba* through retrotransposition and exon shuffling of alcohol dehydrogenase (*ADH*) and *yellow emperor* (*ymp*). (*A*) The MP architecture scenario. (1) Along the branch leading the *dyak*, the MP reconstruction infers the duplication of modules 17143 and 00370 followed by their subsequent merge to form *jgw*. (2) Along branch leading the *dmel*, we see an interaction between modules 17143 and 17258 to form *ymp*. However, there is strong evidence that the gene pairs *GE10684-GE10685* and *GG12235-GG12236* are actually single gene orthologs of *ymp*. These gene break errors lead to an incorrectly inferred ancestral architecture for *ymp* in which the modules 17143 and 17258 are found in separate genes rather than fused in a single gene. (3) *ADH* consists of a single module 00370, and there are multiple copies of this module in isolated form in many genomes. Multiple cases of fusions with *ADH*-derived modules have also been found experimentally, suggesting that *ADH* may be enriched for fusion events. We find module 00370 fused to the architecture 03541-01876 in *dpse*, which has not been previously found in literature. It is possible that *GA25237* and *GA25238* are further examples of *ADH*-derived chimeric genes. (*B*) A genome level view of *ymp* reveals gene break errors in the *ymp* orthologs. This is supported by exon structure and genome alignment and partially supported by EST evidence: *ymp* in *dmel* has multiple full ESTs (e.g. ESTs span entire gene), and *GE10684* and *GE10685* in *dyak* have multiple spanning ESTs, but none of *GG12235* or *GG12236* in *dere* nor *GF17267* in *dana* have ESTs. *ADH* is found on a different chromosome (scaffold). (*C*) The inferred evolutionary history of *jgw* in *dyak*. *ymp* is first duplicated to create a second copy *yande* (Long et al. 2003), then *ADH* is retrotransposed between the third and fourth exons of *yande* followed by degeneration of the *yande* exons found after the insertion point. Exons in gray represent exons that are not part of the longest transcript.

species	# genes	# genes w/ EST (mRNA-seq)	# gene pairs	# gene pairs w/ EST (mRNA-seq)	# gene pairs w/ spanning EST (mRNA-seq)	error rate (%) of EST (mRNA-seq)	# families w/ EST (mRNA-seq)	# families w/ spanning EST (mRNA-seq)	error rate (%) of EST (mRNA-seq)
dmel	4769	4439 (4455)	420	348 (333)	18 (10)	5.17 (3.00)	284 (296)	18 (10)	6.34 (3.38)
dyak	5604	429	806	18	4	22.22	17	4	23.53
dere	5169	1697	715	59	7	11.86	55	7	12.73
dana	4988	1804	655	92	11	11.96	77	11	14.29
dpse	5092	935 (4587)	618	30 (496)	6 (16)	20.00 (3.23)	23 (413)	6 (16)	26.09 (3.87)
dwil	4827	1592	507	69	7	10.14	58	7	12.07
dmoj	4797	1713 (4507)	640	62 (575)	10 (36)	16.13 (6.26)	55 (465)	10 (36)	18.18 (7.74)
dvir	4809	1847	654	81	12	14.81	77	12	15.58
dgri	5227	1777	651	75	3	4.00	64	3	4.69
total	45,282	16,233 (13,549)	5666	834 (1404)	78 (62)	9.35 (4.42)	451 (766)	52 (51)	11.53 (6.66)

Table S1: EST evidence for genes in merge/split families. See Table 1 for details. Similar statistics are also provided across architecture families, where at least one gene pair must be present in the architecture family for it to be included in the count. Large error rates can be attributed to low counts.

	total	consistent		inconsistent		unknown	
		EST	mRNA-seq	EST	mRNA-seq	EST	mRNA-seq
scenarios	3858	583 (15.1%)	619 (16.0%)	48 (1.2%)	41 (1.1%)	3227 (83.6%)	3198 (82.9%)
merges	4876	1130 (23.2%)	1992 (40.9%)	54 (1.1%)	55 (1.1%)	3692 (75.7%)	2829 (58.0%)
splits	5659	1400 (24.7%)	1776 (31.4%)	32 (0.6%)	34 (0.6%)	4227 (74.7%)	3849 (68.0%)

Table S2: EST and mRNA-seq evidence for fusion and fission events. The number of consistent, inconsistent, and unknown scenarios and events based on EST and mRNA-seq evidence. This data excludes the 249 families without merges or splits in the reconstruction.

species	# proteins	# exons	# modules	O	E	O/E	# domains	O	E	O/E
dmel	14080	55172	19224	11810	285	41.49	18278	1405	610	2.30
dyak	16077	58629	20409	13008	337	38.60	18349	1471	604	2.44
dere	15044	55947	19448	12306	311	39.63	18309	1405	642	2.19
dana	15069	56304	17002	10553	205	51.44	18987	1542	698	2.21
dpse	16099	57556	16772	10323	183	56.54	19500	1592	692	2.30
dwil	15512	56273	15803	9754	160	61.10	18995	1537	669	2.30
dmoj	14594	54664	16949	10891	213	51.19	17529	1409	576	2.45
dvir	14491	54760	17258	11069	222	49.85	18026	1468	620	2.37
dgri	14982	56250	17590	11260	224	50.31	18855	1610	673	2.39
total	135948	505555	160455	100974	2138	47.23	166828	13439	5782	2.32

Table S3: Correlation of module boundaries and domain boundaries with exon boundaries. For each species, the total number of proteins, exons, modules, and domains is provided. Furthermore, the observed and expected number of exon-bordering module (the number of modules in which both boundaries are within ±10 aa of an exon) and the fold percentage are provided. To calculate the expected number of exon-bordering modules, we derived the probability P of an exon border falling onto any amino acid by dividing the total number of exon borders by the total length of proteins. We also determined the total number T of amino acids within ±10 aa of any module boundary. Based on a null hypothesis of randomly distributed exon borders, the product PT gives the expected number of exon borders within ±10 aa of a module boundary; thus, the expected number of exon-bordering modules is $E = (PT/\#exons)^2(\#modules)$. The same analysis is performed exchanging module with domains. P-values were calculated based on a chi-square distribution (dof = 1), and all p-values satisfied $p < 2.23 \times 10^{-308}$. Column sums may not equal total due to rounding of the expected value.

species	0-0			1-1			2-2			sym			non-sym		
	O	E	O/E	O	E	O/E	O	E	O/E	O	E	O/E	O	E	O/E
dmel	9560	2071	4.62	67	1226	0.05	23	793	0.03	9650	4089	2.36	2160	7721	0.28
dyak	10779	2315	4.66	48	1340	0.04	23	860	0.03	10850	4515	2.40	2158	8493	0.25
dere	10168	2187	4.65	57	1269	0.04	23	815	0.03	10248	4271	2.40	2058	8035	0.26
dana	8594	1880	4.57	44	1077	0.04	27	705	0.04	8665	3662	2.37	1888	6891	0.27
dpse	8434	1807	4.67	49	1075	0.05	21	692	0.03	8504	3574	2.38	1819	6749	0.27
dwil	7876	1708	4.61	46	1026	0.04	18	645	0.03	7940	3379	2.35	1814	6375	0.28
dmoj	8978	1929	4.65	52	1104	0.05	20	740	0.03	9050	3774	2.40	1841	7117	0.26
dvir	9110	1942	4.69	61	1148	0.05	20	743	0.03	9191	3833	2.40	1878	7236	0.26
dgri	9225	1990	4.64	52	1168	0.04	19	747	0.03	9296	3905	2.38	1964	7355	0.27
total	82724	17829	4.64	476	10435	0.05	194	6739	0.03	83394	35003	2.38	17580	65971	0.27

Table S4: Intron phases of exon-bordering modules. For each species, we list the observed and expected numbers of modules with the given intron-phase combinations, where a module is labeled with the phases of its flanking introns. The expected numbers were calculated as in (Long et al. 2003). Specifically, assuming that any two introns can flank an module, the expected number of modules with intron-phase (i, j) is given by $E_{ij} = P_i P_j N$, where $0 \leq i, j \leq 2$, P_i is the proportion of intron phase i actually observed, and N is the total observed number of intron associations. P-values were also calculated based on a chi-square distribution (dof = 1), and all p-values satisfied $p < 2.23 \times 10^{-308}$. Column sums may not equal total due to rounding of the expected value.

	0-0			1-1			2-2			sym			non-sym		
species	O	E	O/E	O	E	O/E	O	E	O/E	O	E	O/E	O	E	O/E
dmel	668	246	2.71	149	146	1.02	39	94	0.41	856	487	1.76	549	918	0.60
dyak	745	262	2.85	143	152	0.94	31	97	0.32	919	511	1.80	552	960	0.58
dere	693	250	2.78	137	145	0.95	41	93	0.44	871	488	1.79	534	917	0.58
dana	770	275	2.80	139	157	0.88	40	103	0.39	949	535	1.77	593	1007	0.59
dpse	784	279	2.81	164	166	0.99	38	107	0.36	986	551	1.79	606	1041	0.58
dwil	750	269	2.79	149	162	0.92	40	102	0.39	939	533	1.76	598	1004	0.60
dmoj	703	250	2.82	147	143	1.03	44	96	0.46	894	488	1.83	515	921	0.56
dvir	734	258	2.85	139	152	0.91	43	99	0.44	916	508	1.80	552	960	0.58
dgri	809	284	2.84	165	167	0.99	42	107	0.39	1016	558	1.82	594	1052	0.57
total	6656	2372	2.81	1332	1389	0.96	358	897	0.40	8346	4658	1.79	5093	8781	0.58

Table S5: Intron phases of exon-bordering domains. See Table S4 for details. All p-values satisfied $p < 1\times10^{-5}$ except for the 1-1 domains.

Table S6: Lineage-specific merge and split events.

species	dist	M	S	genes[a]	% dist[b]	% M (ratio)[c]	% S (ratio)[c]	% dist$_l$[d]	% M$_l$ (ratio$_l$)[e]	% S$_l$ (ratio$_l$)[e]
dmel	11.23	457	233	446	2.9	9.4 (3.2)	4.1 (1.4)	3.7	16.3 (4.4)	5.2 (1.4)
dyak	8.57	268	634	341	2.2	5.5 (2.5)	11.2 (5.1)	2.8	9.6 (3.4)	14.2 (5.0)
dere	8.57	233	340	341	2.2	4.8 (2.2)	6.0 (2.7)	2.8	8.3 (2.9)	7.6 (2.7)
dana	53.40	317	510	335	13.8	6.5 (0.5)	9.0 (0.7)	17.6	11.3 (0.6)	11.4 (0.6)
dpse	55.80	362	523	319	14.4	7.4 (0.5)	9.2 (0.6)	18.4	12.9 (0.7)	11.7 (0.6)
dwil	62.49	309	540	318	16.1	6.3 (0.4)	9.5 (0.6)	20.6	11.0 (0.5)	12.1 (0.6)
dmoj	32.74	283	546	295	8.4	5.8 (0.7)	9.7 (1.1)	10.8	10.1 (0.9)	12.2 (1.1)
dvir	32.74	301	403	335	8.4	6.2 (0.7)	7.1 (0.8)	10.8	10.8 (1.0)	9.0 (0.8)
dgri	37.11	269	735	314	9.6	5.5 (0.6)	13.0 (1.4)	12.3	9.6 (0.8)	16.5 (1.3)
total	302.65	2799	4464	3044	77.9	57.4 (0.7)	78.9 (1.0)	-	-	-

[a]Number of fused genes for which the split form consists of two adjacent genes.
[b]Branch length divided by the total branch length.
[c]Number of merges or splits in this genome divided by the total number of merges or splits, and ratio of (% M)/(% dist) or (% S)/(% dist).
[d]Branch length divided by the total leaf branch length.
[e]Number of merges or splits in this genome divided by the total number of lineage-specific merges or splits, and ratio of (% M$_l$)/(% dist$_l$) or (% S$_l$)/(% dist$_l$).

rank	GO ID	GO term	k	m	fold	p-value	p-value	q-value
1	GO:0007275	multicellular organismal development	323	1119	1.43	2.24×10^{-13}	2.44×10^{-6}	4.39×10^{-3}
2	GO:0032502	developmental process	355	1253	1.40	2.02×10^{-13}	3.41×10^{-6}	4.39×10^{-3}
3	GO:0032501	multicellular organismal process	382	1358	1.39	6.30×10^{-14}	2.79×10^{-6}	4.39×10^{-3}

Table S7: GO enrichment for genes undergoing module rearrangement in conservative set of architecture families. See Table 2 for details. Here, $n = 2506$ and $N = 12,408$, and we used a p-value cutoff of $p < .01$.

MERGES	all	w/o generation	w/ generation
number of events	2559	1676 (65.5%)	883 (34.5%)
retained at least one split architecture	2392 (93.5%)	1620 (96.7%)	772 (87.4%)
retained both split architectures	1413 (55.2%)	1413 (84.3%)	n/a

SPLITS	all	w/o loss	w/ loss
number of events	2446	670 (27.4%)	1776 (72.6%)
retained merged architecture	465 (19.0%)	443 (66.1%)	22 (1.2%)

Table S8: Retainment of ancestral architectures by merge and split events in conservative set of architecture families. See Table 3 for details.

species	dist	M	S	% dist	% M (ratio)	% S (ratio)	% dist_l	% M_l (ratio_l)	% S_l (ratio_l)
dmel	11.23	174	122	2.9	6.8 (2.3)	4.2 (1.4)	3.7	13.7 (3.7)	5.2 (1.4)
dyak	8.57	110	309	2.2	4.3 (1.9)	10.7 (4.9)	2.8	8.7 (3.1)	13.5 (4.8)
dere	8.57	106	169	2.2	4.1 (1.9)	5.9 (2.7)	2.8	8.4 (3.0)	7.4 (2.6)
dana	53.40	146	271	13.8	5.7 (0.4)	9.4 (0.7)	17.6	11.5 (0.7)	11.9 (0.7)
dpse	55.80	170	248	14.4	6.6 (0.5)	8.6 (0.6)	18.4	13.4 (0.7)	10.9 (0.6)
dwil	62.49	146	293	16.1	5.7 (0.4)	10.2 (0.6)	20.6	11.5 (0.6)	12.8 (0.6)
dmoj	32.74	152	282	8.4	5.9 (0.7)	9.8 (1.2)	10.8	12.0 (1.1)	12.3 (1.1)
dvir	32.74	137	204	8.4	5.3 (0.6)	7.1 (0.8)	10.8	10.8 (1.0)	8.9 (0.8)
dgri	37.11	127	386	9.6	4.9 (0.5)	13.4 (1.4)	12.3	10.0 (0.8)	16.9 (1.4)
total	302.65	2567	2880	77.9	49.4 (0.6)	79.3 (1.0)	-	-	-

Table S9: Lineage-specific merge and split events in conservative set of architecture families. See Table S6 for details
.

chimeric gene	parental genes	source	citation	detected	reason not detected
CG10102	CG12505 (Arc1)	E	Z		only one hit satisfies %id thr
CG12592	CG12819 (sle), CG18545	E	R,Z		under-clustered module instances
CG18853	CG12822, CG11205 (phr)	E	R,Z	X	
CG18217	CG17286 (spd-2), CG4098	E	R,Z	X	
CG31332 (unc-115)	CG31352	E	Z	X	
CG31687	CG31688, CG2508 (cdc23)	E	R,Z	X	
CG33105 (p24-2)	CG33104 (eca), CG31352	E	Z	X	
CG32745	CG3458 (Top3β)	E	Z		CG32745 became psuedogene
CG31864 (Qtzl)	CG5202 (escl), CG12264	E	R,Z		frameshift
CG32744 (Ubi-p5E)	CG11700	E	Z	X	
CG18789	CG18787	E	Z		only one hit
CG17706	CG4211 (nonA)	E	Z		only one hit; CG17706 became CG40282
CG32788 (Crg-1)	CG12632 (fd3F)	N	Z		only one hit
CG41454	CG33217	N	Z		only one hit
CG31054	CG4849	N	Z		only one hit
CG32318	CG9191 (Klp61F), CG9187 (psf1)	N	R,Z	X	
CG32821	CG12655	R	Z	X	
CG40100	CG30022	U	Z	X	
CG33221	CG33213	R	Z		only one hit satisfies %id thr
CG32733	CG9821	E	Z	X	
CG12184	CG12179	U	Z		only one hit satisfies %id thr
CG12824	CG12825	R	Z		no hits
CG14810	CG14811	R	Z		only one hit
CG32584	CG15645 (cerv)	N	Z		CG32583 split into CG42299/CG42300
CG17797 (Acp29AB)	CG15818	E	Z		no hits satisfy %id thr
CG17440	CG17446	R	Z	X	
CG13794	CG33296	N	Z	X	
CG6900	CG6891	R	Z		only one hit
CG14666 (Tim17a2)	CG10090 (Tim17a1)	U	Z	X	
CG32071	CG12522 (Gtp-bp)	R	Z		no hits
CG14628	CG18823	U	Z	X	
CG17472	CG31680	U	Z		only one hit
CG33317	CG3584 (qkr58E-3)	R	Z		CG33317 became pseudogene
CG3410 (lectin-24A)	CG7106 (lectin-28C)	U	Z		no hits satisfy %id thr
CG9902	CG7692	U	Z	X	
CG10991	CG9360	R	Z		no hits satisfy %id thr
CG7804	CG10327 (TBPH)	U	Z		only one hit satisfies %id thr
CG9906	CG11958 (Cnx99A)	R	Z	X	
CG11235	CG11958 (Cnx99A)	N	Z	X	
CG31904	CG13796, CG7216 (Acp1)	E	R	X	
CG30457	CG10953, CG13705	E	R		no hits
CG17196	CG17197, CG17195	E	R		no hits satisfy %id thr
CG11961	CG9416, CG30049	E	R	X	
CG3978 (pnr)	CG9656 (grn), CG10278 (GATAe)	E	R	X	
CG6844 (nAcRα-96Ab)	CG5610 (nAcRα-96Aa), CG11348 (nAcRβ-64B)	E	R	X	
CG6653 (Ugt86De)	CG31002 (Gga), CG17200 (Ugt86Dg)	E	R		overlapping alignments
CG31668	CG33124, CG8451	E	R		overlapping alignments

Table S10: Detection of previously identified chimeric genes. Chimeric sources are divided into 'E' (from exons of another parental gene), 'N' (from intron or intergenic module), 'R' (simple tandem repeats or repetitive elements) and 'U' (unknown sources). Citations are 'R' (Rogers et al. 2009), and 'Z' (Zhou et al. 2008). Hits refer to LALIGN hits for the chimeric gene, e.g. only one LALIGN hit for the chimeric gene satisfied the percent identity threshold.

Online Supplementary Files

1 Analysis of 9 *Drosophila* genomes

`flies9.stree`, `flies9.smap`, `flies9.names.txt` : Phylogeny and species-to-gene map of 9 *Drosophila* species used in our analysis.

`flies9.ids.txt` : Gene names.

`regs.tar.gz` : Modules and module families.

`fams.txt`, `fams.ms.txt`, `fams.ms.cons.txt` : Architecture families, "merge/split" architecture families, and conservative "merge/split" architecture families.

`fams.ms.tar.gz` : For each architecture family, the extant architectures, bootstrapped gene trees, reconstructed architecture scenario, and a figure of the reconstructed architecture scenario.

2 Catalog of genes grouped by mechanism of formation

`nbrs.txt`,`nbrs_genes.txt` : Genes involved in fusion/fission of neighboring genes (either as a parent or child), and whether they are supported by experimental evidence.

`dupmerge.txt` : Genes involved in large-loop mismatch repair or replication slippage.

`retro.txt` : Genes involved in fusion/fission via retrotransposition.

`dupdeg.txt` : Genes involved in duplication-degeneration.

References

Fong J. H, Geer L. Y, Panchenko A. R and Bryant S. H. 2007. Modeling the evolution of protein domain architectures using maximum parsimony. J. Mol. Biol. **366**:307–315.

Gilbert W, Long M, Rosenberg C and Glynias M. 1995. Tests of the exon theory of genes. In M. Go and P. Schimmel, eds., Tracing Biological Evolution in Protein and Gene Structures. Proceedings of the 20th Taniguchi International Symposium, Division Of Biophysics, Elsevier Science.

Hahn M. W, Han M. V and Han S.-G. 2007. Gene family evolution across 12 *Drosophila* genomes. PLoS Genet. **3**:e197–.

Jones C. D and Begun D. J. 2005. Parallel evolution of chimeric fusion genes. Proc. Natl. Acad. Sci. U. S. A. **102**:11373–11378.

Jones C. D, Custer A. W and Begun D. J. 2005. Origin and evolution of a chimeric fusion gene in drosophila subobscura, d. madeirensis and d. guanche. Genetics **170**:207–219.

Kummerfeld S. K and Teichmann S. A. 2005. Relative rates of gene fusion and fission in multi-domain proteins. Trends Genet. **21**:25–30.

Long M. 2001. Evolution of novel genes. Current Opinion in Genetics & Development **11**:673–680.

Long M, Betran E, Thornton K and Wang W. 2003. The origin of new genes: glimpses from the young and old. Nat. Rev. Genet. **4**:865–875.

Long M and Langley C. H. 1993. Natural selection and the origin of jingwei, a chimeric processed functional gene in drosophila. Science **260**:91–95.

Rasmussen M. D and Kellis M. 2007. Accurate gene-tree reconstruction by learning gene- and species-specific substitution rates across multiple complete genomes. Genome Res. **17**:1932–1942.

Rasmussen M. D and Kellis M. 2010. A bayesian approach for fast and accurate gene tree reconstruction. Mol. Biol. Evol. msq189–.

Rogers R. L, Bedford T and Hartl D. L. 2009. Formation and longevity of chimeric and duplicate genes in drosophila melanogaster. Genetics **181**:313–322.

Snel B, Bork P and Huynen M. 2000. Genome evolution: gene fusion versus gene fission. Trends Genet. **16**:9–11.

Young M. D, Wakefield M. J, Smyth G. K and Oshlack A. 2010. Gene ontology analysis for rna-seq: accounting for selection bias. Genome Biol. **11**:R14.

Yu J, Pacifico S, Liu G and Finley R. 2008. Droid: the drosophila interactions database, a comprehensive resource for annotated gene and protein interactions. BMC Genomics **9**:461.

Zhou Q, Zhang G, Zhang Y, Xu S, Zhao R, Zhan Z, Li X, Ding Y, Yang S and Wang W. 2008. On the origin of new genes in drosophila. Genome Res. **18**:1446–1455.